React

Koci Erik

March 26, 2021

1 Props

Le **props** sono utilizzate per passare parametri ad una funzione; le props stesse sono oggetti e non cambiano nel tempo, esse sono **statiche**. L'assegnamento delle props è chiamato **destrutturizzazione**.

```
<NomeComponente nome="Erik"/>
```
Nel caso in cui volessimo attribuire dei valori di **default**, utilizziamo questa sintassi:
```
NomeComponente.defaultProps = { nome:  "Non disponibile" }
```

1.1 React.Fragment

Il tag **<React.Fragment>** è specifico di React e permetti di ovviare al problema di inserimento del tag div all'interno di altri blocchi. Nel casoin cui volessimo per esempio inserire una riga all'interno di una tabella creando un nuovo componente, il compilatore ci darà errore.

1.2 PropTypes

Per aggiungere i controlli sulle proprietà d'ingresso, si sfrutta propTypes valorizzandola con un oggetto. E' necessario importarlo nel file.

```
NomeComponente.propTypes = { nome:  PropTypes.string }
NomeComponente.propTypes = { nome:  PropTypes.number }
```

1.3 Metodo map

La funzione Javascript `map()` ci permette di ottenere un nuovo array da scorrere, costruito sulla base dell'array originario.

```
hooby.map(item => {
   lavoro sul singolo elemento
});
```

Per non sporcare il codice html, possiamo incapsulare il metodo `map` all'interno di un oggetto.

```
const itemJSX = (
   hooby.map(item => {item})
   );
```

A questo punto possiamo inserire nel codice Html il solo ogggetto:

```
return ( { itemJSX } );
```

1.3.1 SetInterval

Funzione javascript che ci permette di creare un timer:

```
setInterval( () => { i++;},1000);
```

Il metodo **render()** viene evocato solo una volta! Quindi nel caso noi volessimo modificare un paragrafo non lo possiamo fare. Per bypassare questo problema utilizzeremo lo **state**.

2 Stato di un componente

La caratteristica dell'oggetto **state** è che non appena viene modificato, la modifica viene trasmessa immediatamente sul DOM. Esso confronta i due DOM aggiornando con le nuove modifiche.
E' necessario importarlo nel file **{useState}**. Per inizializzare una variabile di stato dovrò utilizzare la funzione **useState**.

```
function Stock(props){
  const [prezzo, setPrezzo] = useState(120);
  const [ora, setOra] = useState('16:00');
}
```

3 Gestire gli eventi

L'interazione tra utente e interfaccia è dettata soprattutto dalla proprietà onClick. Qui presente una lista dei **principali eventi** collegati al mouse:

`onClick:` **Click** sull'elemento
`onMouseEnter:` **Mouse** spostato sopra un elemento
`onKeyDown:` **Pressione** di un tasto su tastiera
`onChange:` **Cambio dell'opzione** selezionata dall'utente
`onSubmit:` Quando il form viene **inviato**

4 useContext

Utilizzando **useContext** siamo in grado di rendere delle proprietà **visibili a tutti i componenti figli** senza passare i valori tramite props.

Il valore di useContext è determinato dal tag `<MyContext.Provider>` il quale assumerà le props da rendere disponibili a tutti i figli.
Bisogna ricordarsi inoltre che l'argomento da passare alla funzione useContext deve essere un oggetto! Per creare un `context` dobbiamo utilizzare `createContext`.

$$createContext()$$

$$useContext(MyContext)$$

```
const themes = {
  light: {
    foreground: "#000000",
    background: "#eeeeee"
  },
  dark: {
    foreground: "#ffffff",
    background: "#222222"
  }
};

const ThemeContext = React.createContext(themes.light);

function App() {
  return (
    <ThemeContext.Provider value={themes.dark}>
      <Toolbar />
    </ThemeContext.Provider>
  );
}

function Toolbar(props) {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}

function ThemedButton() {
  const theme = useContext(ThemeContext);
  return (
    <button style={{ background: theme.background, color: theme.foreground }}>
      I am styled by theme context!
    </button>
  );
}
```

Quando un componente chiama **useContext** esso sarà sempre rirenderizzato ogni volta che `value` cambia. In questo esempio potremo usare le proprietà date dal componente **App** nel componente `ThemedButton`, richiamando useContext.

5 Accorgimenti

In presenza di un **if** nel quale abbiamo solo la condizione then, per stampare ad esempio un paragrafo avendo delle condizioni, possiamo utilizzare questa sintassi:

```
{ eta>=18 and and <h3>Sono maggiorenne</h3> }
```

La notazione degli **Hook** equivale alle funzioni.

Per **bloccare** un comportamento predefinito di un tag `html`, si deve richiamare `e.preventDefault`.

5.0.1 React.memo

React.memo è un cosiddetto higher order component (componente di ordine superiore).
Se il componente renderizza lo stesso risultato a partire dalle stesse props, esso si può racchiudere in una chiamata a React.memo per ottenere un miglioramento della performance, in alcuni casi tramite la memoizzazione del risultato. In altre parole, React eviterà di ri-renderizzare il componente, riutilizzando l'ultima renderizzazione.
Questo metodo esiste solamente come **strumento per ottimizzare la performance**. Non utilizzarlo per "prevenire" la renderizzazione, in quanto farlo può essere causa di bug.